STANDARDISATION & GUIDELINES

jmzML, an open-source Java API for mzML, the PSI standard for MS data

Richard G. Côté[1], Florian Reisinger[1]* and Lennart Martens[2,3]*

[1] EMBL Outstation, European Bioinformatics Institute, Wellcome Trust Genome Campus, Cambridge, UK
[2] Department of Medical Protein Research, VIB, Ghent, Belgium
[3] Department of Biochemistry, Ghent University, Ghent, Belgium

We here present jmzML, a Java API for the Proteomics Standards Initiative mzML data standard. Based on the Java Architecture for XML Binding and XPath-based XML indexer random-access XML parser, jmzML can handle arbitrarily large files in minimal memory, allowing easy and efficient processing of mzML files using the Java programming language. jmzML also automatically resolves internal XML references on-the-fly. The library (which includes a viewer) can be downloaded from http://jmzml.googlecode.com.

Received: October 26, 2009
Revised: December 7, 2009
Accepted: January 2, 2010

Keywords:
Bioinformatics / Java API / Proteomics Standards Initiative MS standard / XML processing

Proteomic experiments can generate an exorbitant volume of data. Without the availability of efficient processing and analysis tools, these data will, for all their potential, remain largely unused. In an effort to promote data comparison, exchange and verification, the Human Proteome Organization Proteomics Standards Initiative (PSI) recently officially released the mzML standard for MS data representation [1]. This format merges (and obsoletes) the previous two prominent MS formats mzData [2] and mzXML [3], and adds novel data capture functionality as well. Although several mzML APIs already exist in the C or C++ programming languages (*e.g.* openMS [4] and ProteoWizard [5]; for a full list see http://www.psidev.info/index.php?q = node/257), no full API has yet been made available in pure Java.

We here present jmzML, a mature Java API for mzML files that combines a small memory footprint with a fully functional object model including automatic XML reference resolving without sacrificing the overall speed of data access. Written in 100% pure Java, the jmzML API is also inherently platform independent, and contains a simple yet powerful viewer application.

One of the key features of the mzML specification is the abundance of internal references in the XML document, *e.g.* from a spectrum to the instrument configuration used to acquire that spectrum. mzML allows for the definition of repeated elements such as the abovementioned instrument configuration in a single location, relying on internal references to point out to these elements where required. Additionally, spectra can also refer to each other, as is the case for precursor MS and product MS/MS spectra. This reliance on references limits repetition in the file, thus reducing file size. The corollary, however, is that when reading an mzML file, it is necessary to continuously look up such referenced elements, a task that is typically handled by reading an entire mzML file into memory for fast access to the referenced elements. However, as mzML files can easily exceed several gigabytes in size, the large amount of memory required to read these files fully into memory quickly makes it impossible to process these files on a standard PC or laptop. To circumvent this problem, while still allowing referenced elements to be retrieved on-the-fly, jmzML uses the innovative XPath-based XML indexer (xxindex) component to access the mzML file. xxindex is essentially a random-access XML

Correspondence: Professor Lennart Martens, Department of Medical Protein Research, VIB and Department of Biochemistry, Ghent University, B-9000 Ghent, Belgium
E-mail: lennart.martens@UGent.be
Fax: +32-9-264-94-84

Abbreviations: CV, controlled vocabularies; **PSI,** Proteomics Standards Initiative; **xxindex,** XPath-based XML indexer

*These authors contributed equally to this work.

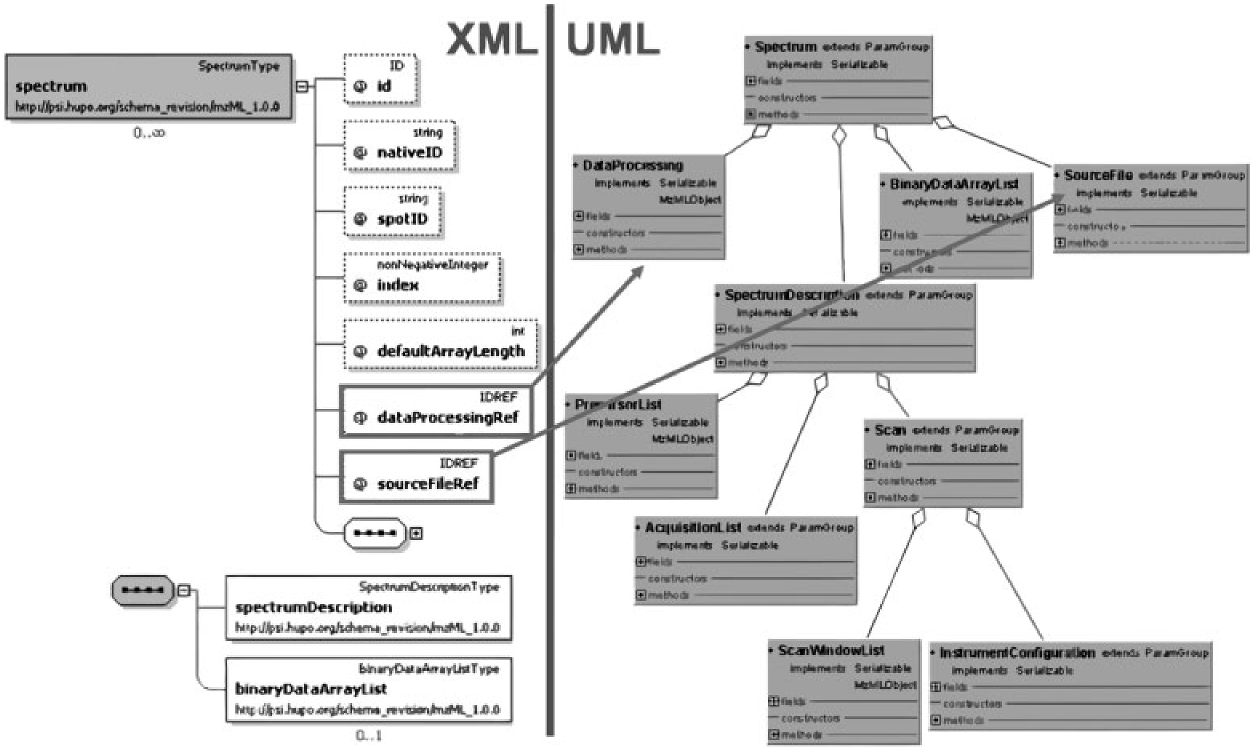

Figure 1. Schematic representation of the automatic reference resolving in jmzML. The left-hand diagram shows a part of the XML schema structure for mzML, which often uses internal references to point to the shared elements, as highlighted in the figure. The Unified Modeling Language (UML) diagram on the right displays the actual Java objects available to a jmzML user. Note that elements that are merely referenced in the XML are directly available as full Java objects in jmzML (arrows), and that this automatic, memory-efficient reference resolving occurs at any depth (lower part of UML diagram on right).

file reader that first indexes a file, and subsequently allows the user to retrieve any XML element based on its XPath. The overall effect for jmzML is that xxindex allows it to treat the whole mzML file as a swap file, obviating the need to load the file into memory. When a jmzML user attempts to retrieve an XML element from the file as one or more Java objects in memory, any internal references will be detected by jmzML and the corresponding XML fragments will be read into memory automatically through xxindex. After unmarshalling the relevant XML fragments into Java objects, jmzML seamlessly provides a complete object model to the user, implicitly replacing internal XML references by the actual objects (Fig. 1). This process can cascade to any depth (*i.e.* elements that contain references that themselves contain referenced elements). As only the relevant portions of the whole XML file are read into memory during this process, the memory footprint remains trivial compared with the size of the original file. By using the live indexing-based random-access capabilities of xxindex, jmzML can handle large files at good performance without the need to parse more than a small section of the file into memory at any one time.

Referenced objects will implicitly be shared across several referent objects. jmzML will recognize this central role of the referenced objects, and will improve the efficiency of the unmarshalling process by caching certain of these referenced objects in memory for prompt retrieval. This caching mechanism is user-configurable and can be tweaked to achieve any desired memory-to-performance balance.

Another specific property of the mzML format is its reliance on controlled vocabularies (CV), through "cvParam" elements in the format. jmzML provides full access to these CV parameters, and actively interprets several, low-level parameters (for instance, to correctly decode binary spectrum or chromatogram information). However, detailed handling of high-level CV parameters (such as sample origin) is left to the user, as this handling will likely differ from application to application. An example of such an end-user application is given by the mzML semantic validator [6], which is built on jmzML.

The jmzML library also comes with a simple but powerful, interactive mzML viewer that can be used to load and view spectra and chromatograms from multiple, very large mzML files simultaneously, illustrating the usefulness of the low-memory footprint achieved by jmzML. Screenshots of this graphical tool are shown in Fig. 2.

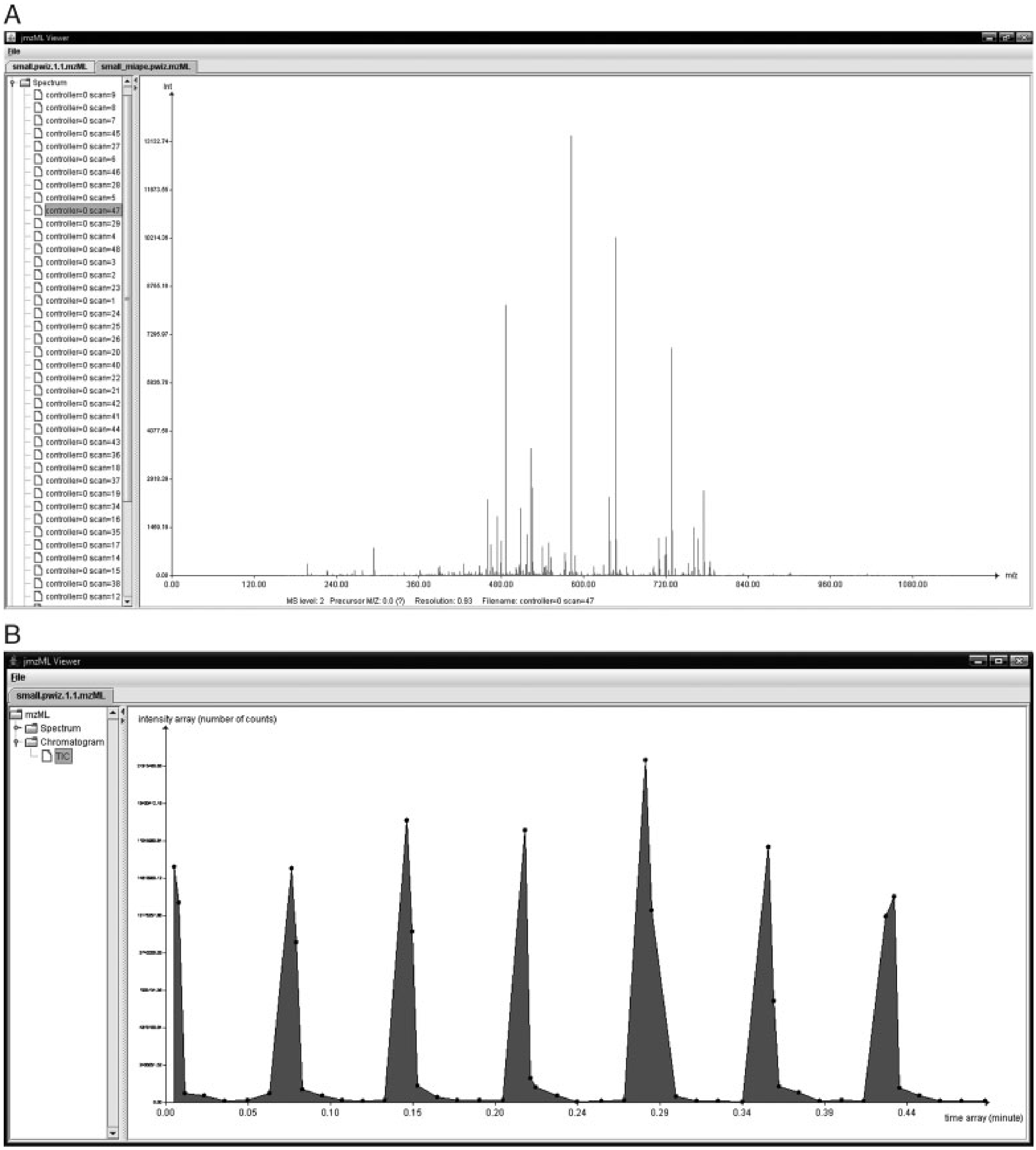

Figure 2. Screenshots of the jmzML spectrum viewer that are included in the library. The top screenshots shows several simultaneously opened mzML files, one *per* tab. The tree on the left provides the list of spectrum and chromatogram identifiers found within each file, and selecting an identifier shows the corresponding MS spectrum or chromatogram on the right. Both the spectrum and chromatogram viewer are interactive, allowing click-and-drag zooming. (A) The spectrum viewer. (B) The chromatogram viewer.

jmzML has already been successfully used as the basis for a PSI-MS mzML semantic validator [6] and will be a cornerstone of the next incarnation of the PRIDE database [7]. jmzML is freely available, and is released as open source under the permissive Apache 2.0 license. The binaries, source code and documentation can be downloaded from the project web site at http://jmzml.googlecode.com.

This work has been supported by the "ProDaC" grant LSHG-CT-2006-036814 of the European Union. The authors thank Henning Hermjakob and Rolf Apweiler for their support. L. M. thanks Joël Vandekerckhove for his support.

The authors have declared no conflict of interest.

References

[1] Deutsch, E., mzML: A single, unifying data format for mass spectrometer output. *Proteomics* 2008, *8*, 2776–2777.

[2] Orchard, S., Hermjakob, H., Julian, R. K., Runte, K. *et al.*, Common interchange standards for proteomics data: public availability of tools and schema. *Proteomics* 2004, *4*, 490–491.

[3] Pedrioli, P. G. A., Eng, J. K., Hubley, R., Vogelzang, M. *et al.*, A common open representation of mass spectrometry data and its application to proteomics research. *Nat. Biotechnol.* 2004, *22*, 1459–1466.

[4] Sturm, M., Bertsch, A., Gröpl, C., Hildebrandt, A. *et al.*, OpenMS – an open-source software framework for mass spectrometry. *BMC Bioinformatics* 2008, *9*, 163.

[5] Kessner, D., Chambers, M., Burke, R., Agus, D., Mallick, P., ProteoWizard: open source software for rapid proteomics tools development. *Bioinformatics* 2008, *24*, 2534–2536.

[6] Montecchi-Palazzi, L., Kerrien, S., Reisinger, F., Aranda, B. *et al.*, The PSI semantic validator: a framework to check MIAPE compliance of proteomics data. *Proteomics* 2009, *9*, 5112–5119.

[7] Martens, L., Hermjakob, H., Jones, P., Adamski, M. *et al.*, PRIDE: the proteomics identifications database. *Proteomics* 2005, *5*, 3537–3545.